HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Daniel Erler

Analyse der Kommunikation zwischen Steuergeräten bei unsicherem Wissen

2010

Fakultät **Mathematik/ Naturwissenschaften/ Informatik**

BACHELORARBEIT

Analyse der Kommunikation zwischen Steuergeräten bei unsicherem Wissen

Autor:
Daniel Erler

Studiengang:
Informatik

Seminargruppe:
IF07w2-B

Erstprüfer:
Prof. Günter Werner

Zweitprüfer:
Dr. Helmut Neemann

Mittweida, 2010

Bibliografische Angaben

Erler, Daniel: Analyse der Kommunikation zwischen Steuergeräten bei unsicherem Wissen, 29 Seiten, 2 Abbildungen, Hochschule Mittweida (FH), Fakultät Mathematik/ Naturwissenschaften/ Informatik

Bachelorarbeit, 2010

Referat

In der vorliegenden Arbeit wird die Kommunikation zwischen Steuergeräten mit Hilfe von Zustandsmodellen analysiert und auf Plausibilitäten beim Schließen von Zuständen eingegangen. Die Steuergerätebeschreibungen wurden dafür in SCXML überführt und weiterverarbeitet. Es handelt sich um keine vorherige Simulation, sondern um eine nachträgliche Analyse. Im Automobil kommunizieren viele Steuergeräte gleichzeitig über CAN-Bus miteinander, die vorher nur in kleinerem Rahmen getestet werden konnten. Inhalt dieser Arbeit ist es, Nutzen aus dabei aufgezeichneten Daten zu ziehen, um nachträglich einzelne Steuergeräte auf Fehlerzustände zu überprüfen. Dabei wird berücksichtigt, dass das Wissen über die Abläufe im Fahrzeug unvollständig ist.

I. Inhaltsverzeichnis

II. Abbildungs- und Tabellenverzeichnis

Abbildungen

Tabellen

III. Vorwort

Es ist möglich Zustandsdiagramme für Steuergeräte zu testen. Es gibt auch Tests für abgeschlossene Teilsysteme im Automobil. Jedoch wurde bislang mehr Wert darauf gelegt, die Komplexität im Vorfeld gering zu halten und nicht das gesamte System zu testen. In diesem Dokument soll gezeigt werden, wie man diese Beschränkung aufheben und das fertige System testen kann. Im Rahmen einer Messdatenauswertung wurden dafür bereits Daten aufgezeichnet, die von verschiedenen Steuergeräten in einem Testfahrzeug über einen gemeinsamen Controller Area Network - Bus (CAN) gesendet wurden. Für jedes dieser Steuergeräte ist eine Zustandsautomatenbeschreibung vorhanden. Die Art der Ausführung der Beschreibung variiert dabei. Die Herausforderung besteht darin, diese Beschreibungen in ein einheitliches Format zu überführen und mit Hilfe dieser Beschreibungen die aufgezeichneten Daten auf Fehlerzustände einzelner Steuergeräte zu untersuchen. Für die IAV ist es interessant zu wissen, ob sich Steuergeräte gemäß ihrer Beschreibung/Spezifikation verhalten haben. Dieser Test erfolgt an Hand der Beschreibung und der aufgezeichneten Messdaten. Dabei gibt es kein automatisiertes Verfahren, diesen Abgleich bei kommunizierenden Steuergeräten durchzuführen.

1 Begriffswelt

1.1 Unsicheres Wissen

Zitat Wikipedia [Wikipedia(2010)]: Als unsicheres Wissen werden in der Künstlichen Intelligenz und Wissensrepräsentation Informationen bezeichnet, die aufgrund mangelnder Genauigkeit oder Verlässlichkeit ungewiss sind.

1.2 Signal und Ereignis

In dieser Arbeit sind mit Steuergerät ausschließlich elektronische Steuergeräte gemeint. Die Kommunikation zwischen den Steuergeräten erfolgt über Signale. Signale sowie Zeitinformationen können durch Ereignisse dargestellt werden. Das Automatenmodell kann somit mit Ereignissen statt Signalen dargestellt werden.

1.3 Automat

Ein Automat kann als Graph dargestellt werden. Der Graph besteht dann aus Knoten (Zuständen) und unidirektional gerichteten Kanten (Zustandsübergänge, Transitionen). An den Kanten werden zusätzlich die Informationen angebracht, welche den Zustandsübergang bewirken (Eingaben und sonstige Schaltbedingungen). Es gibt Akzeptoren und Transduktoren. Akzeptoren lesen die Eingabe und befinden sich danach in einem Zustand. Zustände die als Endzustand gekennzeichnet sind akzeptieren die Eingabe. Bei Transduktoren ist nicht der Zustand Mittelpunkt der Betrachtung sondern Ausgaben und Aktionen die der Automat vollzieht. Bei Mealy-Automaten hängt diese Aktion von Eingaben und Zustand ab, und wird deshalb mit an die Kante geschrieben. Bei Moore-Automaten hängt diese Aktion nur vom Zustand ab, sie wird als Eingangsaktion in den Zustand geschrieben. 1.1(Bildquelle:Wikipedia)

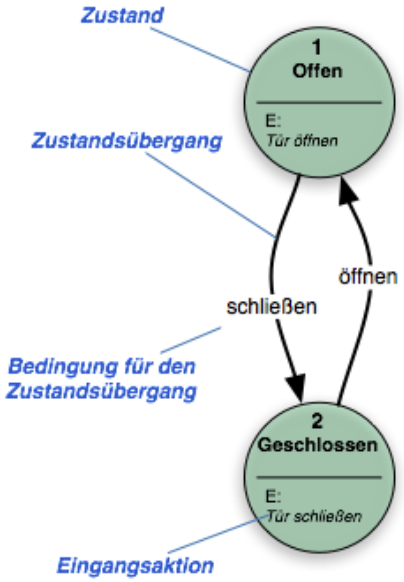


Abbildung 1.1: Automatensymbolik am Beispiel eines Moore-Automaten

Im Folgenden ist mit Automat immer ein endlicher Zustandsautomat vom Typ Transduktor gemeint. Bei Medwedew-Automaten (einem Spezialfall des Moore-Automaten) ist der Zustand auch die Ausgabe. Ob der Automat einen Mealy-Automat, Moore-Automat oder Medwedew-Automat darstellt, ist für die meisten Betrachtungen dieser Arbeit irrelevant. Sollte eine Unterscheidung notwendig sein, so wird darauf verwiesen.

Automaten werden neben der Darstellung als Graph (bei Akzeptoren Übergangs oder Zustandsübergangsdiagramm genannt) auch mittels Übergangstabellen dargestellt. Dies findet vor allem in der Digitaltechnik Anwendung. Dafür werden die alten Zustände und die Eingaben als Bildungsinformation verwendet. Meistens sind auch Kodierungstabellen beigelegt.

Durch die Angabe der Kodierung der Eingaben wird meist auch ausgeschlossen, dass diese gleichzeitig eintreffen. Sollte dies nicht der Fall sein, müssten auch die Übergangsbedingungen so gestaltet werden, dass genau spezifiziert ist, welcher Übergang unter welcher Bedingung genommen wird.

Die Steuergeräte lassen sich durch ereignisgesteuerte, endliche Automaten darstellen. Das bedeutet Eingaben und Ausgaben sind Ereignisse, welche Zustandswechsel in Steuergeräten (betrachtetes, vernetzte Steuergeräte) veranlassen können.

Warum unsicheres Wissen bei Steuergerätekommunikation?

Die vernetzten Steuergeräte senden sich Ereignisse zu, ohne über sichere Kenntnis des Zustands des empfangenden Steuergeräts zu verfügen. Unter bestimmten Umständen ist es jedoch möglich den Zustand eines Steuergeräts zu ermitteln, oder zumindest

Tabelle 1.1: Beispiel einer Zustandsübergangstabelle

Eingabe	Zustand	neuer Zustand	Ausgabe
timeout	liesA	repA	ta
	liesB	repB	tb
	repA	repA	tc
	repB	repB	tc
a	liesA	liesB	gut
	liesB	liesB	wb
	repA	liesB	la
	repB	liesB	ma
b	liesA	liesA	wa
	liesB	liesA	gut
	repA	liesA	mb
	repB	liesA	lb
c	liesA	liesA	l
	liesB	liesB	m
	repA	repA	n
	repB	repB	o

auszuschließen, dass sich ein Steuergerät in einem bestimmten Zustand befindet. Die Steuergeräte müssen in manchen Fällen von bestimmten Zuständen ausgehen (unsicheres Wissen).

1.4 Echtzeit und virtuelle Zeit

Die Zeit kann nicht in allen Steuergeräten gleich gemessen werden. Das Verhalten der Steuergeräte hängt jedoch von der Zeit ab. Die nachträglichen Überprüfungen können nicht in Echtzeit erfolgen, das Verhalten erfolgt jedoch in dieser.

Die virtuelle Zeit ist ein Hilfsmittel, dass die Bindung an bestimmte Zeitpunkte aufhebt. Damit ist eine Betrachtung im Nachhinein möglich, mit welcher Überprüfungen durchgeführt werden können, die in Echtzeit nicht möglich wären. Die Einhaltung virtueller Zeit wird durch die Nutzung von Zeitstempeln und das Verarbeiten der Ereignisse in bestimmter Reihenfolge implementiert.

1.5 Plausibilisierung

Das Verhalten der Steuergeräte wird plausibilisiert. Dafür wird das aufgezeichnete Verhalten der Steuergeräte gegen ein Modell des Soll-Verhaltens geprüft. Signale die von Steuergeräten gesendet wurden, werden in Ereignissen wiedergegeben. In dieser Arbeit wird des Weiteren der Begriff Plausibilität als quantitatives Maß für die Qualität (im Sinne von Güte) verwendet. Die Plausibilität nimmt Werte von 0 bis 1 an, wobei 0 gar

nicht plausibel und 1 total plausibel darstellt. Messwerte können bei SCXML durch Ereignisse dargestellt werden, oder in einem sogenannten Datenmodell direkt als Daten hinterlegt werden. Auch eine Aufnahme eines Messwertes in das Datenmodell und spätere Anpassung dieses Wertes durch Ereignisse ist möglich. Zeit kann z.B. als Wert im Datenmodell zur Verfügung gestellt werden, aber Ereignisse können vergangene Zeit symbolisieren.

Eine Plausibilitätskontrolle welche alle korrekten Fälle als plausibel und alle inkorrekten Fälle als nicht plausibel bewertet, ist nicht entwerfbar. Plausibilisierungen welche alle Fehlerfälle erkennen, werden (auf Grund der Unsicherheit des Wissens) unweigerlich auch ein paar korrekte Verhalten als unplausibel bewerten. Plausibilisierungen welche alle korrekten Verhalten als solche erkennen, werden auch kleinere Abweichungen vom geforderten Verhalten als plausibel akzeptieren.

Da es sich „nur“ um eine Plausibilitätskontrolle handelt sollten keine korrekten Verhalten als unplausibel deklariert werden. Deswegen empfiehlt sich eine zahlenmäßige Wichtung der Plausibilität.

2 Anforderungen

2.1 Allgemeine Anforderungen

Bei dem gewählten Automaten handelt es sich um einen ereignisgesteuerten, endlichen Automaten.
Es gibt zwei Grundszenarien die unterschiedliche Anforderungen stellen.

Szenario1:
Die Verwendung vereinfachter Beschreibungen ohne spezifizierten Fehlerzuständen ist Ziel dieses Szenarios. Als Eingabe dient eine Maschinenbeschreibung, die Ausgabe ist eine um Fehlerzustände erweiterte Maschine.

Eingabe:	Maschine(nbeschreibung)
Ausgabe:	um Fehlerzustände erweiterte Maschine

Szenario2:
Ziel dieses Szenarios ist die Analyse der Steuergerätekommunikation.

Eingabe:	Maschinenbeschreibung in SCXML
Testlauf:	Ausgabe bei fehlgeschlagenen Tests
Abfrage aktueller Zustände:	Jederzeit, Problem mit automatischen Übergängen

In dieser Arbeit wird das Augenmerk auf das zweite Szenario gelegt. Daraus ergeben sich folgende grundlegende Anforderungen an die Funktionalität.

- Eine Überführung der Beschreibung in ein Modell.
- Eine Rücküberführung des Modells in eine Beschreibung.
- Das Modell beschreibt den Automaten treffend.
- Eine Engine gibt das Zustandsbild der Maschine wieder (die Engine „lebt“)

2.2 Dynamik

Der grundlegende Ablauf einer Plausibilitätskontrolle umfasst dann folgende Schritte:

- SCXML-Beschreibung(en) erstellen
- SCXML-Beschreibung(en) einlesen
- SCXML-Baum in Java-Objekten speichern
- Einlesen zu validierender Messergebnisse (was wurde kommuniziert)

- Plausibilität für nächstes Ergebnis, nächste Ergebnisse ausrechnen
- Stellen die unplausibel sind oder einen definierten Alarmzustand beschreiben anzeigen

2.3 Klassen-Modell

Verfeinerung Zustandsmodellsicht:
(Automatenbeschreibung, Modellelemente, statische Sicht)

Zustände	Punkte die während der Ausführung eines Automaten erreicht werden können, unterschiedliche Automaten haben unterschiedliche Zustände Verfeinerung der Zustände selbst ist möglich. Um dies zu ordnen, wird eine Hierarchie Zwischenschicht eingeschoben (Nebenläufigkeit)
Ereignisse	sehr abstrakt, als Eingabe sowie Ausgabe verwendbar, kann Transitionen auslösen, muss es aber nicht
Transitionen	beschreiben den Übergang zwischen den Ausführungspunkten (Zuständen) eines Automaten, nur innerhalb des selben Automaten möglich (können dabei aber auch Ereignisse auslösen, die für andere Automaten relevant sind)

Verfeinerung Laufzeitsicht/Laufzeitelemente (dynamische Sicht):
(Automatenart/Automatenexemplar) - Veränderbarkeit der Daten

aktuellZustand	an einen Zustand gebunden, mit Zeitinformationen, Plausibilität verbunden
Pfadelement	alter Zustand (mit Plausibilität) Zeitpunkt des Verlassen Übergang (mit Plausibilität) Zeitpunkt des Betreten Zustand (mit Plausibilität)
Pfad	Pfadelemente

Zustände sind Teil des Automatenmodells, da sie nur in genau einer Automatenart (welche durch das Modell beschrieben wird) genau einmal vorkommen können.
Zustandsübergänge (Transitionen) sind ebenfalls Teil des Modells, da sie nur Zustände in genau einer Automatenart verbinden können.
Ereignisse können in verschiedensten Transitionen Anwendung finden, sie sind nicht nur Bestandteil genau eines Automaten.

Als Analysemodell dient ein Zustandsautomatenmodell. Zur Analyse werden die Steuergeräte in eine einheitliche Zustandsbeschreibung gebracht. Dieses Modell wird in Da-

teiform abgelegt. Als Formatspezifikation wurde SCXML (gemäß [(W3C)(2010)]) gewählt.

Probleme:

- SCXML-Framework ist für Simulation gebaut und nutzt daher Echtzeit
- wählt nur einen Weg, statt aller Wege, daher erst recht keine Plausibilitäten

FSFO-FSM würde zum Beispiel alle Wege beschreiten, allerdings ist FSFO-FSM nur ein Konzept.

- neuer Executor
- Rückschritt zu Digester und bauen neuen Parser für neue Objekte
- stärkere Working Draft-Orientierung um genau spezifiziertes Verhalten zu haben
- Weglassen zu komplexer Implementierungen wie XPATH-Profil

Grund für diese Änderungen ist die tiefere Verankerung von Plausibilitäten und der Wegfall von Zwischenumwandlungen (Parser von Commons SCXML umbiegen manche sonst nicht geparste (W3C-Spezifikation konforme) Datei zu parsen, Executor diese auch verstehen lassen, eigenes Erweiterungsformat ankoppeln)

2.4 Statik

Die statische Struktur wurde in XML festgehalten. Diese orientiert sich stark an dem SCXML Working Draft. Es gibt SCXML als Wurzelelement, es gibt Übergänge (Transitionen) und es gibt Zustände. Es gibt Vordefinitionen für Verfeinerungen bei Zuständen. (Initial) Bei Zuständen gibt es jene mit parallelen Teilen (Parallel), jene mit sequentiellen (State) und jene, die nicht aus Teilzuständen zusammengesetzt sind (Final). Die genaue Struktur kann in [(W3C)(2010)] nachgelesen werden.

3 Grundlagen

Dieses Kapitel stellt einen grundlegenden Überblick über existierende Mittel und Frameworks dar. Im letzten Abschnitt werden Vorteile und Nachteile tabellarisch gegenübergestellt.

3.1 Automaten in UML

UML nahm die von David Harel vorgeschlagene Idee von Hierarchie bei Zustandsdiagrammen in den Standard auf. UML bietet noch weitere Arten von Zustandsdiagrammen an (z.B. mit Variablen erweiterte Automaten die bei jedem Übergang ihre Variablen ändern können). Deswegen hat sich für diese Art der Zustandsdiagramme der Begriff Harel Statechart etabliert. Die Idee der Hierarchie findet sich auch in den nachfolgend vorgestellten Modellen wieder.

3.2 Hierarchie bei Automaten

Um dem Betrachter einen verständlichen Überblick über den Automaten zu geben, wurde es im Lauf der Zeit mit der zunehmenden Anzahl an Zuständen notwendig, sich Alternativen auszudenken, um das Modell weiterhin strukturiert darzustellen. Eine der Maßnahmen in dem Zusammenhang war die Einführung von Hierarchie. Wie die Hierarchie aufgebaut wird, ist von Konzept zu Konzept unterschiedlich.

3.3 SCXML

SCXML (State Chart XML) ist zur Zeit (Mai 2010) ein Working Draft des World Wide Web Consortium (W3C). Die W3C ist eine Organisation die weltweit Mitglieder hat, die an der Ausarbeitung von Standards wie SCXML arbeiten. Als Working Draft werden Standards bezeichnet, die sich noch in der Entwurfsphase befinden. Die aktuellste Spezifikation findet sich im Internet. [1]

SCXML ist als XML Notation von Harel Statecharts geeignet.

Da SCXML als verallgemeinerter Nachfolger die CCXML (Call Control XML) Spezifikation ersetzen soll, konzentriert es sich auch hauptsächlich auf Sprachverarbeitung. SCXML ist jedoch auch für die Beschreibung komplexer Automaten, die nichts mit

[1] http://www.w3.org/TR/scxml/

Sprachverarbeitung zu tun haben gedacht. SCXML bietet bereits Anbindungen an Laufzeitumgebungen wie zum Beispiel bei Apache Commons. [2]

Die Verwendung dieser erwies sich jedoch im Test als noch unpraktikabel. So sind in dieser Java Implementierung Funktionen als „deprecated“ also veraltet bezeichnet und dennoch sind diese zum Teil unverzichtbar. Als Standardbeispiele zu SCXML dienen eine Mikrowelle für die Veranschaulichung der XML-Struktur und eine Stoppuhr für die Verarbeitung mit dem Apache Commons Framework. Doch auch die Stoppuhr-Implementierung ist unsauber. Hätte das Beispiel Teilzustände oder Nebenläufigkeit besessen, wäre es so nicht implementierbar.

3.4 EHA

EHA (Extended Hierarchical Automata) bestehen aus sequentiellen Automaten, einer Verfeinerungsfunktion zum Aufbau einer Hierarchie und Ereignissen zum Steuern der Übergänge.

Die sequentiellen Automaten bilden ein paarweise disjunktes Mengensystem. Ein sequentieller Automat besteht aus Zuständen und den Beziehungen dieser untereinander. Die Verfeinerungsfunktion der EHA baut eine baumartige Hierarchie auf. Es gibt genau einen sequentiellen Automaten, der als Wurzel dient. Alle anderen sequentiellen Automaten haben genau einen Elternknoten. Der Aufbau ist zyklenfrei (siehe auch [Latella et al.(1999)Latella, Majzik, and Massink] S. 335).

Nehmen wir an es würden im Ausgangsautomaten die Zustände s_1, s_2 und s_3 existieren und s_1 sowie s_3 würden Subzustände (aber keine nebenläufigen) besitzen. Somit würden drei sequentielle Automaten nötig sein. Ein Automat der die Beziehungen der Zustände in s_1 repräsentiert, ein weiterer für die Beziehungen in s_3 und einer der die Beziehungen zwischen s_1, s_2 und s_3 definiert. Die Wurzel würde dann der Automat mit den Beziehungen zwischen s_1, s_2 und s_3 bilden.

Der sequentielle Automat, welcher die Beziehungen der Zustände in s_1 repräsentiert, würde das einzige Element der Verfeinerungsmenge von s_1 sein. Die Verfeinerung von s_2 wäre eine leere Menge. Die Verfeinerung von s_3 beinhaltet dann den Automaten mit den Beziehungen der Zustände in s_3 als einziges Element. Nebenläufigkeit in Zuständen würde durch mehrere Elemente in der Verfeinerungsmenge ausgedrückt werden.

[2] http://commons.apache.org/scxml/

3.5 HMSM

Eine HMSM (Hierarchical Multi State Machine) besteht aus einer endlichen Menge an Zuständen (S) und einer endlichen Menge an Übergängen (Transitionen).

Zur Beschreibung wurden in [Gabrielian and Iyer(1992)] die Begriffe „Markierung“ und „feuernde Transition“ aufgegriffen. Diese finden vorwiegend in Petrinetzen Verwendung.

Eine Transition die ausgeführt wird feuert. (Selten wird hiervon abweichend definiert: Die Transition wird gefeuert.)

Zu einer Zeit t ordne eine Abbildung jedem Zustand wahr oder falsch zu, respektive ob dieser eingetreten ist oder nicht. Diese Abbildung wurde „marking“ genannt.

Definition 3.1 „marking“ zur Zeit t:

$$m_t : S \rightarrow \{false, true\}$$

Dies kann „Markieren“ heißen. Folgerungen aus der Begriffswelt der Petrinetze wären dann:

- Zustände die zu einer Zeit t eingetreten (aktiv) sind, heißen markiert zur Zeit t
- Die Gesamtheit aller markierten Zustände zur Zeit t heißt Markierung zur Zeit t

Da in [Gabrielian and Iyer(1992)] aber mehrere „marking“ zu einer „marking sequence“ zusammengefasst wurden, beschreibt der Begriff dort sowohl die Abbildungsfunktion als auch das „positive“ Ergebnis dieser (Markierung).

Eine Folge von markierenden Funktionen, ließe sich sehr wohl definieren, aber um die Begriffe klarer zu gestalten, wird sich hier (unabhängig der Intention von [Gabrielian and Iyer(1992)]) mit dem Begriff Markierung stets auf das „positive“ Ergebnis der Abbildung bezogen.

Definition 3.2 Markierung zur Zeit t:
M_t ist die Gesamtheit aller Zustände für welche die Abbildungsfunktion Def. 3.1 zur Zeit t true liefert.

Dabei wird Zeit als diskret betrachtet. t ist außerdem relativ zum aktuellen Zeitpunkt zu betrachten. M_5 ist dann zum Beispiel die Markierung, die in fünf Zeiteinheiten eintreten wird.

Eine HMSM kann mehrere Zustände zur selben Zeit annehmen. Ob eine Transition deterministisch oder nichtdeterministisch ist wird durch TIL (Temporary Interval Logic)

Regeln festgelegt und ist unabhängig von allen anderen Transitionen. TIL-Regeln definieren zeitliche Bedingungen wie z.B: Wenn Zustand Z seit n Zeiteinheiten gilt.
Die Überprüfung ob eine Transition ausgelöst wird, wird bei den HMSM unterteilt in die drei Teile: Vorbedingungen, Kontrollbedingungen und Konsequenzen.

Alle drei Teile werden jedoch durch TIL beeinflusst.

Als Vorbedingung wird hier eine bestimmte erforderliche Zustandskonstellation angesehen.

Ist diese Zustandskonstellation eingetreten, so werden die Kontrollbedingungen überprüft. Die Kontrollbedingungen liefern immer einen Wahrheitswert. Ist dieser Wert wahr, dann werden die Konsequenzen eingeleitet. Sonst tritt diese Zustandsänderung nicht ein.

3.6 FFA

Unter den FFA (Fuzzy Fuzzy Automata) sind Automaten als Akzeptor zu verstehen. Für die Validierung der Steuergeräte werden jedoch Transduktoren benötigt. Das Modell baut auf Unschärfe auf, wobei viel Wert auf Namen-Wert-Zuordnung gelegt wurde.

Die hier genannte Beschreibung, dient nur zur Vorstellung eines Konzepts und erhebt keinen Anspruch darauf, die genauen Intentionen der ursprünglichen Autoren [Mizumoto and Tanaka(1976)] wiederzugeben.

Als Grundlage dienen Mengen von Fuzzy-Werten. Ein Fuzzy-Wert ist dabei ein Wert aus $[0,1]$. Damit lassen sich Wahrscheinlichkeiten recht gut einbeziehen. Wobei 0 unmöglich und 1 sicher darstellen. Diese Mengen von Fuzzy-Werten finden zweimal Verwendung. Heiße eine dieser Mengen A und eine andere Menge B. Mit $a \in A$ und $b \in B$. Eine konkrete Zuordnung eines $b \in B$ zu einem $a \in A$ werde als 2-Tupel (a,b) geschrieben. Ein „fuzzy Grade“ ist jetzt eine Menge solcher 2-Tupel. In einem „fuzzy Grade“ können auch mehrere (n) Werte aus B einem a zugeordnet werden. Es sind dann einfach zusätzliche Tupel im „fuzzy Grade“.

Bsp.:
$\{(0.3/0.3),(0.7,0.4),(1.0/0.5),(0.7/0.6),(0.3/0.7)\}$

Es gibt verbale Operatoren wie „Sehr“ bzw. „Nicht“, die sich durch ein quadrieren bzw. negieren auswirken. Gerade quadrieren mag für „Sehr“ auf den ersten Blick unlogisch erscheinen. Da quadrierte Wahrscheinlichkeiten immer kleiner sind als unquadrierte. Es mag also im ersten Moment unsinnig erscheinen, ist aber durch Verwendung von zwei unscharfen Mengen durchaus logisch begründbar. Der zweite Wert bleibt immer

erhalten und durch das quadrieren des anderen Wertes wirkt es wie eine schnellere Annäherung an extreme Werte.

Bsp.:

$\{(0.9/0.9),(1.0/1.0)\} \rightarrow \{(0.81/0.9),(1.0/1.0)\}$ schnelleres Wachstum.
$\{(0.9/0.1),(1.0/0.0)\} \rightarrow \{(0.81/0.01),(1.0/0.0)\}$ schnelleres Abflachen.

3.7 FSFO-FSM

Die FSMs (Finite State Machines) vom Typ FSFO (Fuzzy State Fuzzy Output) verwenden Unschärfe um ihre Zustände bzw. die Ausgaben die produziert werden darzustellen. Dadurch können mehrere Zustände zusammengefasst und die Anzahl der Zustände reduziert werden. Das gleiche gilt auch für die Eingaben und Ausgaben. Durch diese Umwandlung ist es auch möglich infinite exakte Beschreibungen (egal ob Eingabe/Ausgabe oder Zustand) auf finite unscharfe abzubilden. Andererseits ist es jedoch nicht gewährleistet eine gegebene FSFO-FSM auch immer als FSM ohne Unschärfe darstellen zu können. Allgemeine Umwandlungen denen FSFO-FSM zu Grunde liegen müssen demnach entweder auch Unschärfe verwenden oder die eigentliche Komplexität abstrahieren.

3.8 Lizenzmodell

Wichtig ist die freie Verwendung auch zu kommerziellen Zwecken. Dieses Kriterium muss bei allen geprüften Verfahren, API's und ähnlichem gegeben sein. Die SCXML-Spezifikation des W3C erfordert die Nennung des W3C, erfüllt aber dieses Kriterium [3]. Das SCXML-Framework bei Apache Commons ist unter der Apache Commons License lizensiert [4], die ebenfalls eine Nennung von Apache Commons erfordert, aber die Kriterien erfüllt. Für HMSM, EHA, FFA, FSFO-FSM sind keine Spezifikationen oder Frameworks vorhanden deren Lizenzbedingungen geprüft werden müssten.

3.9 Eignungsmatrix

Tabelle 3.1 ist eine kurze Gegenüberstellung der bereits unterstützten Funktionalität der Mittel und Frameworks.

Die Nebenläufigkeit in Zuständen wird in SCXML, und EHA durch echte Hierarchie und in HMSM durch Hierarchie und TIL-Regeln geregelt. Zeitabhängigkeiten werden

[3] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
[4] (http://www.apache.org/licenses/LICENSE-2.0)(http://www.apache.org/legal/3party.html)

Tabelle 3.1: Eignungsmatrix

	SCXML	HMSM	FFA	FSFO-FSM	EHA
Nebenläufigkeit in Zuständen	ja	ja	nein	nein	ja
Zeitabhängige Vorgänge	zum teil	ja	nein	nein	nein
Unterstützung von Unschärfe	nein	nein	ja	ja	nein
Lizenzmodell	ok	ok	ok	ok	ok

in SCXML und HMSM unterstützt. Unschärfe wird in FFA und FSFO-FSM unterstützt.

4 Plausibilität

Es gibt viele denkbare Ansätze um Plausibilitäten zu bestimmen. Jeder Ansatz kann dabei nur Näherung sein.

Einfache Strategie:

- nur das dominant Plausible ist plausibel
- Gefahr der Fehlinterpretation

Geplante Strategie:

- multiplikative Pfadplausibilität, mit Aufwertung der Plausibilität, wenn die letzten X Plausibilitäten im Pfad, die Dominantplausibilitäten waren.
- Problem Fuzzy-Grenzen, Typen von Ereignissen, Ereignisklassen, die unterschiedlich aktualisiert werden

4.1 Abhängigkeiten der Plausibilitätsberechnung

Zum einen ist die Berechnung der Plausibilität von den aktuellen Zuständen abhängig, noch mehr Aufschluss gibt allerdings die übertragene Information. Zum Absenden von Ereignissen muss sich das sendende Steuergerät in einem bestimmten Zustand befinden. In anderen Zuständen würde es diese gar nicht senden.

Ein weiterer Einfluss ist die Zeit. Es gibt Bedingungen die eintreten, wenn ein Ereignis in einem bestimmten Zeitintervall eintrifft. Die Grenzen müssen hierbei unscharf betrachtet werden, da Abweichungen bei der Zeitmessung eintreten können. Diese Abweichungen könnten sonst dazu führen, Ereignisse außerhalb des Zeitintervalls als innerhalb anzusehen. Ebenso könnten innerhalb des Zeitintervalls liegende Ereignisse als außerhalb betrachtet werden. Eintreffende Ereignisse werden nur von bestimmten Zuständen verarbeitet. In einem Zustand s sei eine Transition t definiert, die durch ein Ereignis e ausgelöst werden kann. Der Automat habe eine Plausibilität größer 0, dass er sich in Zustand s befinde, als Ereignis e eintrifft. Wird daraufhin eine Reaktion festgestellt, die äußerlich der Transition t entspricht, so ist das ein weiteres Indiz, dass der Automat sich in s befand und die Plausibilität kann (wenn sie nicht bereits maximal ist) nach oben korrigiert werden.

4.2 Relative und absolute Plausibilität

Die Plausibilitätsberechnung kann für jeden Zustand einzeln erfolgen, dabei werden nur Aussagen über die Plausibilität dieses Zustands getroffen und Einflüsse anderer Zustände weggelassen, die Berechnung wird jedoch leichter (relative Plausibilität). Diese Art der Plausibilität sagt aus, wie notwendig es ist, dass man diesen Zustand als eingetreten betrachten muss. Wenn gar nichts über den Automaten bekannt ist, würde hier jeder Zustand die höchste Wahrscheinlichkeit erhalten. Als Grenzen für die niedrigste und höchste Plausibilität bieten sich hierbei 0 (nicht notwendig) bzw. 1 (unter keinen Umständen weglassen) an.

Um eine Aussage treffen zu können, wie wahrscheinlich es ist, dass ein bestimmter Zustand tatsächlich eingetroffen ist, müssen die Wahrscheinlichkeiten aller Zustände verglichen werden. Dieser Vergleich kann sehr kompliziert bis unmöglich sein und ist der Nachteil dieser Art der Plausibilitätsberechnung.

Vorteil:	schneller Überblick über die Gesamtplausibilität des Systems
Nachteil:	schlechte Vergleichbarkeit der Zustände untereinander

Komplizierter ist die Berechnung wenn man die Plausibilität auf alle Zustände bezieht. Die Gesamtplausibilität muss dabei immer 1 sein (absolute Plausibilität). Der Vorteil hier ist, dass die Vergleichbarkeit jederzeit gegeben ist und die zwar immer noch komplizierten Berechnungen etwas leichter werden als die nachträglichen über die relative Plausibilität. Für die Plausibilitätsberechnung wurde die relative Plausibilität gewählt um

Vorteil:	gute Vergleichbarkeit der Zustände untereinander
Nachteil:	anpassen der Teilplausibilitäten aller Zustände bei jeder Änderung nötig, komplizierte Berechnung, Rundungsfehler

die Nachteile der absoluten Plausibilität zu vermeiden.

Maximalplausibilität und Minimalplausibilität

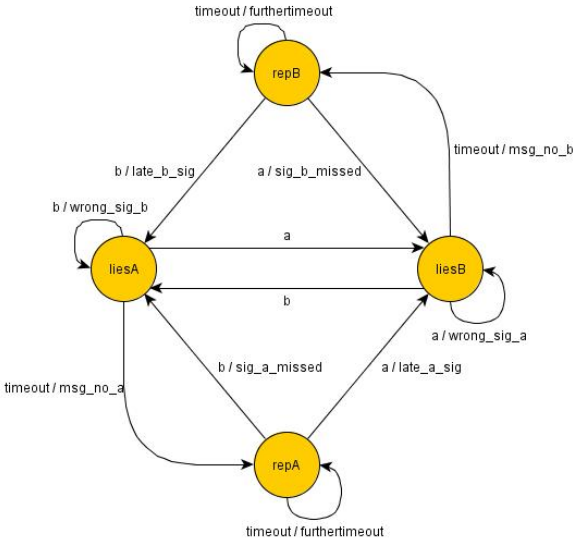
Bei der absoluten Plausibilität ist es möglich die Maximalplausibilität bei jedem Zustand als 1 betrachten, es ist aber auch denkbar die Maximalplausibilität etwas niedriger anzusetzen und die Restplausibilität auf alle anderen Zustände zu verteilen.

Der umgekehrte Einsatz einer Minimalplausibilität, aus der sich dann logischerweise die Maximalplausibilität ergibt ist ebenfalls denkbar.

In dieser Arbeit wird eine Maximalplausibilität von 1 (und damit Minimalplausibilität von 0) verwendet, um sehr unwahrscheinliche Zustände nicht mehr betrachten zu müssen.

4.3 Berechnungsstrategien

Abbildung 4.1: Zeitproblem



Wenn zum Zeitpunkt des Eintreffens eines erwarteten Ereignisses (hier Signal) mittels externer Zeitmessung eine Zeit gemessen wird, die innerhalb des erwarteten Zeitintervalls liegt, gibt man dem passenden Übergang zunächst die höchste Plausibilität. Dennoch muss der Fall, dass das Zeitintervall nicht getroffen wird (i.A. müssen andere Fälle) weiter betrachtet werden. Passen nachfolgende Reaktionen besser auf den Fall, dass das Steuergerät das Ereignis nicht im Zeitintervall entgegengenommen hat, so sollte auch davon ausgegangen werden und die Plausibilität nachträglich angepasst werden. Mit Abbildung 4.1 lässt sich dieses Beispiel darstellen. Sei der Automat in „liesA“ und erhalte Ereignis B gerade noch vor dem Ablauf der Wartezeit („timeout“). Dann könnte die Plausibilität des Zustandes „liesA“ (die der Einfachheit halber hier 1 sei) mit 0.55 multipliziert als Plausibilität für die Transition von „liesA“ zu „liesB“ dienen, die Plausibilität für die Transition von „liesA“ zu „repA“ mit 0.44 etwas niedriger und die Plausibilität für die Transition von „liesA“zu sich selbst (welche ein abgewandeltes Signal benötigt) mit 0.01 sehr gering ausfallen. Damit ergeben sich deutlich geringere Plausibilitäten für die folgenden Zustände. Diese könnte man mittels Addition aller Plausibilitäten die diesen Zustand beinhalten wieder relativieren. Damit verliert man jedoch die Information, welchen Weg man zu diesem Zustand hatte, was zum Beispiel für die Erkennung von genauen Wechselzeitpunkten nicht förderlich ist. Eine andere Variante wäre eine nachträgliche Anpassung der „Geschichte“ des Zustandes. Dabei können die unplausibelsten Wege fallen gelassen werden. Werden die für einen Weg typischen Reaktionen festgestellt und für die anderen Wege nicht, so macht dies den Weg deutlich plausibler. Eine Kombination beider Verfahren ist auch denkbar, so würden die ursprünglichen Bewertungen bewahrt, um als Geschichte des Weges zu dienen und für die Neuberechnung der Wahrscheinlichkeit würden sowohl die plausibelsten Wege als auch die

Summe aller Wege in den Zustand betrachtet werden. Für das Beispiel aus Abbildung 4.1 sind Erweiterungen denkbar. So z.B., dass alle Zustände eine Transition zu sich selbst besitzen, in deren Abarbeitung ein weiteres Ereignis c verarbeitet wird, mit oder ohne sichtbare Reaktion nach außen. Auch bei den aufgezeigten Reaktionen ist denkbar, dass diese nach außen hin nicht sichtbar sind. Bei nicht sichtbaren Reaktionen darf die Plausibilität natürlich nicht nachträglich geändert werden.

Beispiel: Ereignis A trifft zu Ende des vorgeschriebenen Zeitintervall ein.

Tabelle 4.1: A als zu spät erkannt

Zustand	Erkenntnis	Zwischenschritt	Bewertung	Endzustand
liesA	A zu spät	repA	ok	liesB
liesB	B zu spät	repB	A statt B	liesB
repA	A zu spät	repA	ok	liesB
repB	B zu spät	repB	A statt B	liesB

Tabelle 4.2: A als passend erkannt

Zustand	Erkenntnis	Zwischenschritt	Bewertung	Endzustand
liesA	-	liesA	ok	liesB
liesB	-	liesB	A statt B	liesB
repA	-	repA	ok	liesB
repB	-	repB	A statt B	liesB

Tabelle 4.3: timeOutA deutlich größer als timeOutB

Zustand	Erkenntnis	Zwischenschritt	Bewertung	Endzustand
liesA	A zu spät	repA	ok	liesB
liesB	B zu spät	repB	wdh. zu spät, A statt B	liesB
repA	A zu spät	repA	ok	liesB
repB	B zu spät	repB	wdh. zu spät, A statt B	liesB

Tabelle 4.4: timeOutB deutlich größer als timeOutA

Zustand	Erkenntnis	Zwischenschritt	Bewertung	Endzustand
liesA	A zu spät	repA	ok	liesB
liesB	A statt B	liesA	ok	liesB
repA	A zu spät	repA	ok	liesB
repB	A statt B	liesA	ok	liesB

Fazit:
In dem konstruierten Beispiel 4.1 ist der Zustand nach Eintreffen von Ereignis A auf einen einzigen einschränkbar. Wann dieser Zustand erreicht wurde ist jedoch unklar. Fest steht jedoch, dass alle anderen Zustände unplausibel sind und eine Meldung auslösen müssen. Das Beispiel war hier so konstruiert, dass sich der Zustand immer am

Tabelle 4.5: Plausibilitäten aus Reihenfolge gefolgert

	ReihenfolgeAB	ReihenfolgeBA
A nach B	0	1
A kurz nach B	0.4	0.6
B kurz nach A	0.6	0.4
B nach A	1	0

zuletzt eingetroffenen Ereignis orientiert. Sollten die Ereignisse A und B fast gleichzeitig eintreffen reicht dies nicht, da die Reihenfolge des Eintreffens dann nicht sicher feststeht. In dem Fall müssen beide Reihenfolgen getrennt betrachtet werden und die zeitliche Differenz sollte die Plausibilität beeinflussen. Die in 4.5 angegebenen Zahlenwerte sind nicht als verbindlich zu verstehen, sie dienen nur der weiteren Erläuterung des Beispiels.

5 Implementierung

In diesem Kapitel wird die Entwicklung und der Aufbau eines Programms beschrieben, welches auf

5.1 SCXML und Apache Commons

Auf Grundlage eines Working Drafts des W3C ([(W3C)(2010)]) und eines Frameworks von Apache Commons (ebenfalls SCXML) wurde ein Programm geschrieben, welches Zustandsbeschreibungen in SCXML Dateien hält und den zugehörigen Automaten ablaufen lassen kann. Dabei können Ereignisse von außen einfließen um den Ablauf zu steuern. Diese Ereignisse können wiederum einem Eingabestrom entnommen werden und es kann festgestellt werden, zu welchem Zeitpunkt die Zustandsmaschine sich mit Sicherheit in Zuständen aus einer Zustandsmenge befindet, beziehungsweise eine vorgegebene Zustandsmenge verlassen hat. Dieses Wissen wird dazu genutzt um Fehlerzustände zu signalisieren.

Die hierfür aus einer XML-Struktur zu entnehmenden Daten sind sehr komplex und werden mit Hilfe des Digester Frameworks, ebenfalls von Apache Commons, verarbeitet. Das auf Java basierende Framework ermöglicht es Regeln für eigene Suchmuster zu definieren. Die Suchmuster sind an XPath angelehnt. Damit ist es möglich sowohl Objekte als auch Methoden an beliebige Teile der XML-Struktur zu binden. Es können Objekte beliebiger Klassen angegeben werden. Selbst die Kenntnis des Objekttyps ist nicht mehr notwendig, solange die Methode existiert. Dies kann unter anderem durch die Verwendung von Interfaces erreicht werden. Sowohl die Suchmuster als auch die übereinstimmenden Regeln werden auf Stapeln (Stack-objekten) verarbeitet. Damit sind Beziehungen zwischen den einzelnen Objekten modellierbar.

Apache Commons SCXML greift in seiner Implementierung ebenfalls auf das beschriebene Digester Framework zu. Weite Teile des SCXML-Frameworks sind allerdings weder konform mit der aktuellen Spezifikation des W3C, noch mit dieser ersetzbar. Dies liegt daran, dass in Commons SCXML nicht überall Schnittstellen (interface) deklariert wurden und Implementierungen zum Teil als endgültig (final) deklariert worden sind.

5.2 Aufbau und Pakete

Um dennoch die Vorteile des SCXML Frameworks nutzen zu können, wurden daher der Parser, der Executor und große Teile der Implementierung ersetzt. Ein Interface Target (Ziel) verwaltet die Zustände des Automaten, ähnlich wie das im Framework vor-

handene TransitionTarget (Übergangsziel). Jedoch ist dieses nicht direkt implementiert, sondern um abstrakte Klassen erweitert. Eine dieser zwei abstrakten Klassen unterstützt Hierarchie (AbstractNestableTarget), die andere die Besonderheiten eines nicht weiter verfeinerbaren Zustands (AbstractAtomicTarget). Um die Übersicht zu wahren, wurden die Klassen in Pakete aufgeteilt. Zum einen die (nahezu) statisch vorliegenden Daten im Modell (model), zum anderen die ständig Änderungen unterliegenden Daten in der Laufzeitumgebung (runtime). Unter Modell wurden die Klassen nach Typ in Pakete unterteilt: Schnittstellen unter interfaces, Implementierungen in implementations, abstrakte Klassen in abstractClasses.

Es gibt für die verschiedenen Ausdrucksarten verschiedene Evaluatoren, diese wurden auf Grund der Vielzahl in einem eigenen Paket (evaluators) eingeordnet. Auch hier sind Schnittstellen definiert. So existieren Schnittstellen für Bedingungsevaluatoren, Wertevaluatoren und Evaluatoren für Ortsausdrücke (z.B. Evaluation zu URL oder zu Knoten im XML-Baum). Das W3C sieht des Weiteren Profile vor, welche jeweils ihre eigenen Evaluatoren definieren sollen. Die Implementierung aller vorgeschlagenen Evaluatoren (drei Profile mit jeweils drei Evaluatoren) wurde für die eigentliche Arbeit als nicht notwendig erachtet, sollte durch die Schnittstellen aber leicht nachzuarbeiten sein.

Bei den Implementierungen wurde die Profilstruktur als Pakethierarchie angewendet. Dabei wird zwischen Kernfunktionalität (coreModule), DatenModul, (dataModule), externe Kommunikation (externalCommunicationsModule) sowie Script-Funktionalität (script-Module) unterschieden.

5.3 Übergänge

Es gibt viele Möglichkeiten wie ein Übergang zwischen zwei Zuständen stattfinden kann. Um die beste Strategie der Behandlung zu finden seien im Folgenden Unterscheidungsmerkmale genannt, die eine unterschiedliche Behandlung erfordern können. Für die folgenden Betrachtungen werden Beziehungen zwischen den Zuständen durch Verwandtschaftsbegriffe ausgedrückt. Die Teile aus denen ein Zustand besteht (die Verfeinerung des Zustands) sind Kinder dieses (zu verfeinernden) Zustands. Wenn ein Zustand nicht weiter verfeinert werden kann, so heißt dieser atomar.

1. Direkte Übergangsursache

- Immer auszuführender Übergang
- Übergang erfordert Ereignis
- Übergang erfordert sonstige Bedingung
- Übergang erfordert Ereignis und sonstige Bedingung

2. Indirekte Übergangsursache

- Ein Ereignis von aussen trifft ein.
- Ein vorheriger Übergang ändert eine Bedingung.
- Eine explizite Anweisung ein Ereignis zu erzeugen wurde durchgeführt.
- Die Abarbeitung eines Teils des Automaten ist abgeschlossen und erzeugt ein Beendet-Ereignis. (startet ausserdem Verallgemeinerungsprozess)
- Übergang als Teil eines Verfeinerungsprozesses.
- Übergang als Teil eines Verallgemeinerungsprozesses.

3. Beziehung des Ziels des Übergangs zum Ursprung des Übergangs

- Ziel ist ein Nachkomme des Ursprungs.
- Ziel ist ein Vorfahre des Ursprungs.
- Keine Ahnenbeziehung zwischen Ziel und Ursprung.

4. Behandlung von Teilübergängen

- Übergang verlässt einen atomaren Zustand
- Übergang verlässt einen Zustand mit mehreren betretenen Teilzuständen
- Übergang betritt einen atomaren Zustand
- Übergang betritt einen Zustand mit mehreren betretbaren Teilzuständen
- Angabe mehrerer Teilzielzustände

5.3.1 Direkte Übergangsursache

Dieser Teilaspekt beeinflusst das Filtern der passenden Übergänge und die Selektion eines Übergangs aus allen möglichen. Die direkte Übergangsursache ist recht schnell für alle Übergangsarten nach diesem Unterscheidungskriterium gleich zu erfüllen. Dazu muss nur die Menge aller verfügbaren Übergänge schrittweise gefiltert werden. Ist Ereignis oder Bedingung bei dem definierten Übergang nicht angegeben, so erfüllt der Übergang die Filtervoraussetzungen und wird nicht herausgefiltert. Fehlt dem Auslöser der Überprüfung ein Ereignis, so werden alle Übergänge welche Ereignisse erfordern herausgefiltert. Bei den verbliebenen Übergängen wird die sonstige Bedingung überprüft. Ist keine angegeben, gilt es als erfüllte Bedingung und der Übergang muss nicht herausgefiltert werden. Von den verbliebenen Übergängen wird die erste ausgeführt.

5.3.2 Indirekte Übergangsursache

Die indirekte Übergangsursache sagt aus, welche Übergänge für eine Filterung gemäß direkter Übergangsursache überhaupt in Frage kommen. Bei Ereignissen von außen muss sichergestellt sein, dass der Automat ohne das Ereignis von außen keine Übergänge die nicht durch vergangene Zeit ausgelöst werden mehr durchführen könnte.

Aus dem Wissen der zu diesem Zeitpunkt gültigen (Teil)-Zustände, kann die Liste aller möglichen Übergänge erstellt werden. Dabei werden auch die verallgemeinerten Zustände beachtet, jedoch erst nach den genauen (Teil)-Zuständen. Unter der Annahme, dass nur ein Beendet-Ereignis gleichzeitig ausgelöst werden kann, ist auch hier die Liste aller möglichen Übergänge von genauen Teilzuständen zu verallgemeinerten (Teil)-Zuständen erstellbar. Schwieriger wird es bei Verallgemeinerungsprozess, Verfeinerungsprozess oder expliziter Ereigniserzeugung. Hier spielt eine Rolle in welchem Kontext der Vorgang stattfindet.

5.3.3 Ursprung-Ziel-Beziehung

Die Beziehung zwischen Ursprung und Ziel des Übergangs haben Auswirkungen auf die Durchführung des Übergangs. Wenn das Ziel des Übergangs ein Nachkomme des Ursprungs ist, wird in der Durchführung die standardmäßige Verfeinerung gegen den angegebenen Übergang einfach ausgetauscht. Die Verallgemeinerung (wenn das Ziel also ein Vorfahre des Ursprungs ist) muss dagegen alle Teilübergänge mitbeachten.

5.3.4 Teilübergänge

Wenn der Übergang aus einem atomaren Zustand, in einen atomaren Zustand erfolgt, wird der wenigste Aufwand benötigt. Vorausgesetzt der zu betretende Zustand ist kein Nachkomme des Ursprungszustandes werden ausgehend vom Ursprungszustand bis zum gemeinsamen Vorfahren alle Teilzustände verlassen (verallgemeinern) und anschließend zum Zielzustand alle Teilzustände betreten (verfeinern). Sollte der Zielzustand ein Nachfahre des Ursprungszustands sein, so wird nur verfeinert. Wenn der Übergang aus einem atomaren Zustand in einen nicht atomaren Zustand erfolgt, so wird zuerst so verfahren, als sei dieser ein atomarer Zustand und anschliessend wird der Zielzustand verfeinert. Beim Verlassen eines Zustands mit mehreren betretenen Teilzuständen müssen alle Teilzustände hin zum „speziellsten gemeinsamen Vorfahren“ (auch LCA, Last Common Ancestor, Lowest Common Ancestor) verlassen werden und anschließend muss die Verfeinerung wie beim Verlassen eines atomaren Zustand bereits beschrieben erfolgen.

Es ist auch möglich mehrere Ziele anzugeben. Die angegebenen Ziele müssen dabei Teil eines gemeinsamen Zieles sein. Bis zu diesem (LCA der Zielzustände) müssen alle Teilzustände verlassen werden, falls dieser ein Vorfahre ist. Falls dieser ein Nachfahre ist muss bis zu diesem verfeinert werden. Andernfalls muss bis zum LCA von Ausgangszustand und diesem (LCA der Zielzustände) verallgemeinert und in den LCA der Zielzustände verfeinert werden. Von da ab, kann in die angegebenen Zustände gewechselt werden.

Bei Aufteilungen in Parallel-Objekten muss entschieden werden, wo die Verfeinerung

zuerst stattfindet. Wenn eine Verfeinerung an einer Stelle nicht beendet werden kann, so muss der Programmfluss an der Stelle unterbrochen werden und die Verfeinerung der anderen Teilzustände erfolgen. (Eine Verfeinerung kann nicht beendet werden, wenn nach anderen Teilzuständen in dem Parallel-Objekt gefragt wird.) Sollte eine Ringbeziehung auftreten (d.h. die Verfeinerung kann nie abgeschlossen werden)(muss/sollte) ein Fehlerereignis erzeugt werden.

Durchgeführter Ansatz: Die Kindelemente sagen ihrem Eltern-Objekt wie weit sie verfeinert sind, wenn alle Kindelemente sagen komplett, so ist die Verfeinerung abgeschlossen. Wenn alle Kindelemente sagen wir sind genauso weit wie vorher verfeinert, so kann die Verfeinerung nie abgeschlossen werden. Bei allen anderen Meldungen der Kindelemente wird die Verfeinerung fortgesetzt.

Gemäß W3C gibt es einen „final“-Zustand, der selbst nicht weiter verfeinert werden kann, keine ausgehenden Transitionen besitzt, und dessen Betreten seinen Vorfahren durch ein Ereignis signalisiert wird. Dieses Ereignis kann von anderen aktiven Teilzuständen oder Vorfahren genutzt werden, um einen oder mehrere Zustandsübergange auszulösen.

Eine Verfeinerung bei solch einem „final“-Element ändert also die Richtung hin zur Verallgemeinerung. Damit alle parallelen Aktivitäten nachvollzogen werden, muss dabei die Verfeinerung an dieser Stelle als abgeschlossen bezeichnet werden. Danach müssen alle anderen laufenden Verfeinerungen fertig verfeinert werden und erst dann kann die Verallgemeinerung ausgehend von den final-Elementen gestartet werden. Dies kann zum Beispiel durch eine als letzten Schritt eingefügte Kontrolle im Verfeinerungsprozess ob man sich in „final“-Zuständen befindet geschehen.

Literaturverzeichnis

[Gabrielian and Iyer(1992)] A. Gabrielian and R. Iyer. Verifying Properties of HMS Machine Specifications of Real-Time Systems. In *Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 421 – 431. Springer Berlin / Heidelberg, 1992.
DOI: 10.1007/3-540-55179-4_39.

[Latella et al.(1999)Latella, Majzik, and Massink] Diego Latella, Istvan Majzik, and Mieke Massink. Towards a formal operational semantics of UML statechart diagrams. In Paolo Ciancarini, Alessandro Fantechi, and Robert Gorrieri, editors, *Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, volume 139 of *IFIP Conference Proceedings*, pages 331 – 347. Kluwer Academic Publishers, 1999.
ISBN: 0-7923-8429-6 DOI: 10.1.1.21.3906.

[Mizumoto and Tanaka(1976)] M. Mizumoto and K. Tanaka. Fuzzy Fuzzy Automata, 1976.

[(W3C)(2010)] World Wide Web Consortium (W3C). State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Working Draft 13 May 2010. Technical report, World Wide Web Consortium (W3C), 2010.

[Wikipedia(2010)] Wikipedia. Stichwort Unsicheres Wissen, 2010. Online-Quelle.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, 22. November 2010